CPS510

Final Report

University Course Management System

Cesar Perez 500882780
Anthony Greco 500903364

Table of Contents

1. INTRODUCTION

1.1 APPLICATION DESCRIPTION

We have chosen to create a university course management application. There will be three types of users that can access the database: students, teachers, and application admins. All types of users must login with a given username and password to access the application.

Students have the ability to view available courses, enroll/drop courses, view grades, view course history, and view enrolled courses. Teachers have the ability to add/remove courses, add/remove students from their course, and add individual grades to students. Admins have the ability to view the total number of students/teachers, add new teachers/admins, and change any information within the database.

1.2 DATABASE DESCRIPTION

The database holds Student and Course entities. A Student entity is created for every student registered in the system. Students enrolled into the university would be entered into the system automatically. Each Student entity holds multiple attributes, such as name, age, student ID, email, degree, year of study, courses taken, GPA, and so on, to keep track of the student's basic information. The Student entity also holds multiple entities, such as multiple Tuition and Assignment entities. The Tuition entity keeps track of the tuition source—that is, the thing that the student is paying the tuition for—and the amount due, the due date, and whether the student has already paid the tuition. The Mark entity keeps track of the marks that the student has received for any given assignment, and as such, it keeps track of the assignment that the mark was received for, as well as the actual mark received.

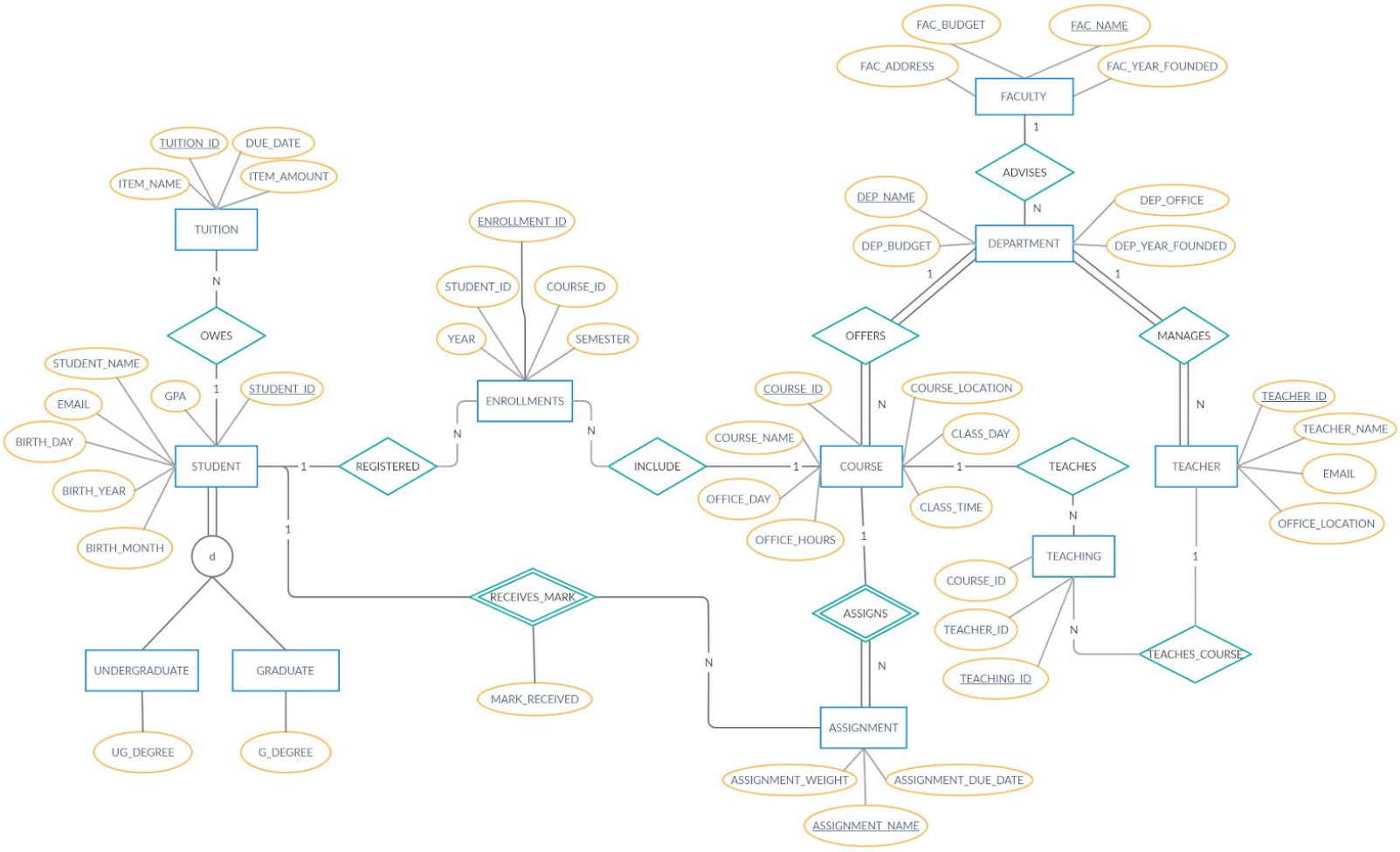
A Course entity is created for every course the university offers. It has several attributes to keep track of the course information, such as course name, professor, meeting times and locations, instruction hours, and so on. Each Course entity holds multiple Assignment entities, one per assignment registered in the course. The Assignment entity keeps track of the basic assignment information, such as assignment name, due date, and weight on the course GPA. At the end, to calculate the student GPA,

the system sums the product of each mark multiplied by the assignment weight, and then divides it by the total number of courses taken by the student.

1.3 ENTITIES AND RELATIONSHIPS

Entities	Relationships
Student	Independent entity. Holds Tuition and Mark entities.
Undergraduate	Subclass of Student.
Graduate	Subclass of Student.
Teacher	Independent entity.
Course	Independent entity. Holds Assignment entities.
Department	Independent entity. Holds Course and Teacher entities.
Faculty	Independent entity. Holds Department entities.
Tuition	Dependent entity to Student. Multiple Tuition entities would be created per tuition source, such as courses, clubs, and other required payments that the student is required to make.
Assignment	Dependent entity to Course. Multiple Assignment entities would be created to represent each assignment that a course has.

2. ER DIAGRAM


FAC_BUDGET
FAC_NAME
FAC_ADDRESS
FAC_YEAR_FOUNDED
FACULTY
1
ADVISES
N
DEP_NAME
DEP_OFFICE
DEP_BUDGET
DEPARTMENT
DEP_YEAR_FOUNDED
1
1
OFFERS
MANAGES
TUITION_ID
DUE_DATE
ITEM_NAME
ITEM_AMOUNT
TUITION
N
OWES
1
STUDENT_NAME
GPA
STUDENT_ID
EMAIL
BIRTH_DAY
STUDENT
BIRTH_YEAR
BIRTH_MONTH
d
1
REGISTERED
N
ENROLLMENT_ID
STUDENT_ID
COURSE_ID
YEAR
SEMESTER
ENROLLMENTS
N
INCLUDE
1
COURSE_ID
COURSE_LOCATION
COURSE_NAME
CLASS_DAY
COURSE
OFFICE_DAY
CLASS_TIME
OFFICE_HOURS
N
1
1
TEACHES
N
TEACHING
COURSE_ID
TEACHER_ID
TEACHING_ID
N
TEACHES_COURSE
1
TEACHER_ID
TEACHER_NAME
EMAIL
TEACHER
OFFICE_LOCATION
1
RECEIVES_MARK
MARK_RECEIVED
N
ASSIGNS
N
ASSIGNMENT
ASSIGNMENT_WEIGHT
ASSIGNMENT_DUE_DATE
ASSIGNMENT_NAME
UNDERGRADUATE
GRADUATE
UG_DEGREE
G_DEGREE

3. CREATING TABLES

```
CREATE TABLE department (
    dep_ID              VARCHAR2(10) PRIMARY KEY,
    dep_name            VARCHAR2(30) DEFAULT 'GENERAL',
    office_location     VARCHAR2(30)
);

CREATE TABLE faculty (
    fac_name            VARCHAR2(30)  PRIMARY KEY,
    fac_budget          NUMBER,
    fac_address         VARCHAR2(30),
    fac_year_founded    VARCHAR2(30) DEFAULT '1980'
);

CREATE TABLE advises (
    fac_name    VARCHAR2(30) REFERENCES faculty(fac_name),
    dep_ID      VARCHAR2(10) REFERENCES department(dep_ID)
);

CREATE TABLE teacher (
    teacher_ID      VARCHAR2(10) PRIMARY KEY,
    teacher_name    VARCHAR2(40) NOT NULL,
    email           VARCHAR2(30) NOT NULL UNIQUE,
    office_location VARCHAR2(30) DEFAULT 'HUB 101'
);

CREATE TABLE course (
    course_ID       VARCHAR2(30) PRIMARY KEY,
    course_name     VARCHAR2(30) NOT NULL UNIQUE,
    course_location VARCHAR2(30),
    office_day      VARCHAR2(10),
    office_hours    VARCHAR2(30),
    class_day       VARCHAR2(10),
    class_time      VARCHAR2(30)
);

CREATE TABLE student (
    student_ID      VARCHAR2(10) PRIMARY KEY,
    birth_date      DATE NOT NULL,
    email           VARCHAR2(30) NOT NULL UNIQUE,
    student_name    VARCHAR2(40) NOT NULL,
    gpa             NUMBER
);
```

```
CREATE TABLE undergraduate (
    student_ID      VARCHAR2(10) PRIMARY KEY REFERENCES student(student_ID),
    birth_date      DATE NOT NULL,
    email           VARCHAR2(30) NOT NULL UNIQUE,
    student_name       VARCHAR2(40) NOT NULL,
    gpa             NUMBER,
    ug_degree       VARCHAR2(30)
);

CREATE TABLE graduate (
    student_ID      VARCHAR2(10) PRIMARY KEY REFERENCES student(student_ID),
    birth_date      DATE NOT NULL,
    email           VARCHAR2(30) NOT NULL UNIQUE,
    student_name       VARCHAR2(40) NOT NULL,
    gpa             NUMBER,
    g_degree        VARCHAR2(30)
);

CREATE TABLE tuition (
    student_ID      VARCHAR2(10) REFERENCES student(student_ID),
    tuition_ID      VARCHAR2(10) PRIMARY KEY,
    item_name       VARCHAR2(30),
    due_date        DATE,
    item_amount     NUMBER NOT NULL
);

CREATE TABLE owes (
    student_ID  VARCHAR2(10) REFERENCES student(student_ID),
    tuition_ID   VARCHAR2(30) REFERENCES tuition(tuition_ID)
);


CREATE TABLE assignment (
    assignment_name     VARCHAR2(50) PRIMARY KEY,
    assignment_weight   NUMBER NOT NULL,
    assignment_due_date DATE
);

CREATE TABLE assigns (
    course_ID       VARCHAR2(30) REFERENCES course(course_ID),
    assignment_name VARCHAR2(50) REFERENCES assignment(assignment_name)
);
```

```
CREATE TABLE receives_mark (
    student_ID      VARCHAR2(10) REFERENCES student(student_ID),
    assignment_name VARCHAR2(50) REFERENCES assignment(assignment_name),
    mark_received   NUMBER NOT NULL
);

CREATE TABLE manages (
    dep_ID      VARCHAR2(10) REFERENCES department(dep_ID),
    teacher_ID  VARCHAR2(10) REFERENCES teacher(teacher_ID),
    PRIMARY KEY (dep_ID, teacher_ID)
);

CREATE TABLE offers (
    dep_ID      VARCHAR2(10) REFERENCES department(dep_ID),
    course_ID   VARCHAR2(30) REFERENCES course(course_ID),
    PRIMARY KEY (dep_ID, course_ID)
);

CREATE TABLE teaching (
    teaching_ID VARCHAR2(10) PRIMARY KEY,
    teacher_ID  VARCHAR2(10) REFERENCES teacher(teacher_ID),
    course_ID   VARCHAR2(30) REFERENCES course(course_ID)
);

CREATE TABLE enrollments (
    enrollment_ID   VARCHAR2(10) PRIMARY KEY,
    student_ID      VARCHAR2(10) REFERENCES student(student_ID),
    course_ID       VARCHAR2(30) REFERENCES course(course_ID),
    year            VARCHAR(10),
    semester        CHAR(2)
);
```

4. CREATING VIEWS

```
/* 9. Display all teachers that teach a computer science course (starts with CPS)
OR an art course (starts with ART) */
CREATE VIEW cs_teachers AS
(SELECT ts.teacher_ID, teacher_name, course_ID
 FROM teaching ts, teacher tr
 WHERE course_ID LIKE 'CPS%' AND
       ts.teacher_ID = tr.teacher_ID)
UNION
(SELECT ts.teacher_ID, teacher_name, course_ID
 FROM teaching ts, teacher tr
 WHERE course_ID LIKE 'ART%' AND
       ts.teacher_ID = tr.teacher_ID)
ORDER BY course_ID;

/* 10. Display count of students, courses, teachers, departments, and assignments
*/
CREATE VIEW table_counts AS
SELECT COUNT(DISTINCT student_id) AS Number_of_Students, COUNT(DISTINCT course_id)
AS Number_of_Courses, COUNT(DISTINCT teacher_id) AS Number_of_Teachers,
        COUNT(DISTINCT dep_ID) AS Number_of_Departments, COUNT(DISTINCT
assignment_name) AS Number_of_Assignments
 FROM student, teacher, course, department, assignment
 WITH READ ONLY;

/* 11. For each student, list their student ID, student name, email, and calculate
the total amount of their tuition */
CREATE VIEW total_tuitions AS
SELECT s.student_id, student_name, email, SUM(item_amount) AS Total_Tuition
FROM student s, tuition t
WHERE s.student_id = t.student_id
GROUP BY s.student_id, student_name, email
ORDER BY student_name
WITH READ ONLY;
```

5. SIMPLE QUERIES AND ADVANCED QUERIES

```
/* 1. List all the students that qualify for the Dean's List */
SELECT student_name AS Deans_List, gpa
FROM student
WHERE gpa >= 3.8
ORDER BY gpa;

/* 2. List all the students (student name and student ID) that are enrolled in
ART201 */
SELECT student_name AS ART201_Students, s.student_id
FROM student s, enrollments e
WHERE course_id = 'ART201'
      AND s.student_id = e.student_id
ORDER BY student_name;

/* 3. Show the average GPA of all students */
SELECT 'The average GPA of all students is: ' AS Average_GPA, AVG(gpa) AS GPA
FROM student;

/* 4. For each course, list the course ID, course name, number of students in the
course, and the number of teachers in the course */
SELECT e.course_id, c.course_name, COUNT(DISTINCT e.student_id) AS
Number_of_Students, COUNT(DISTINCT t.teacher_id) AS Number_of_Teachers
FROM enrollments e, teaching t, course c
WHERE e.course_id = t.course_id
      AND e.course_id = c.course_id
      AND t.course_id = c.course_id
GROUP BY e.course_id, c.course_name
ORDER BY c.course_name;

/* 5. List the average mark, minimum mark, and maximum mark for each assignment in
ART201 */
SELECT a.assignment_name, AVG(mark_received) AS Average_Mark, MIN(mark_received)
AS Lowest_Mark, MAX(mark_received) AS Highest_Mark
FROM assigns a, receives_mark r
WHERE course_id = 'ART201'
      AND a.assignment_name = r.assignment_name
GROUP BY a.assignment_name;

/* 6. For each course, list the course ID and course name, and list all the
assignments in that course and list the average mark, minimum mark, and maximum
mark for each assignment */
SELECT a.course_id, course_name, a.assignment_name, AVG(mark_received) AS
Average_Mark, MIN(mark_received) AS Lowest_Mark, MAX(mark_received) AS
Highest_Mark
```

```
FROM assigns a, course c, receives_mark r
WHERE a.course_id = c.course_id
      AND a.assignment_name = r.assignment_name
GROUP BY a.course_id, course_name, a.assignment_name
ORDER BY course_name;
```

/* 7. List all the teacher ID's, teacher names, teacher emails, and list all the classes they teach and the number of students in those classes */

```
SELECT tr.teacher_id, tr.teacher_name, tr.email, ts.course_id, COUNT(DISTINCT
e.student_id) AS Number_of_Students
FROM teacher tr, teaching ts, enrollments e
WHERE tr.teacher_id = ts.teacher_id
      AND ts.course_id = e.course_id
GROUP BY tr.teacher_id, tr.teacher_name, tr.email, ts.course_id
ORDER BY tr.teacher_name;
```

/* 8. For every department, list the department id, department name, office location, number of teachers, number of courses, and number of students in each department */

```
SELECT d.dep_id, dep_name, office_location, COUNT(DISTINCT teacher_id) AS
Number_of_Teachers, COUNT(DISTINCT o.course_id) AS Number_of_Courses,
COUNT(DISTINCT e.student_id) AS Number_of_Students
FROM department d, manages m, offers o, enrollments e
WHERE d.dep_id = m.dep_id
      AND d.dep_id = o.dep_id
      AND o.course_id = e.course_id
GROUP BY d.dep_id, dep_name, office_location
ORDER BY dep_name;
```

/* 12. Displays all teachers that are teaching a course, alongside the course that they are teaching. */

```
SELECT Teacher.teacher_name, Course.course_name
FROM ((teaching
JOIN Teacher ON teaching.teacher_id = Teacher.teacher_id)
JOIN Course ON teaching.course_id = Course.course_id)
ORDER BY Teacher.teacher_name;
```

/* 13. List all teachers that are not from the science department */

```
SELECT teacher_id, teacher_name, email, office_location FROM teacher
MINUS
(SELECT m.teacher_id, t.teacher_name, t.email, t.office_location FROM manages m,
teacher t WHERE dep_id = '740937523');
```

6. FUNCTIONAL DEPENDENCIES

6.1 Functional Dependencies for Entities

a) **Student Entity**
   - student_ID → birth_date, email, student_name, gpa, birth_date

b) **Course Entity**
   - course_ID → course_name, course_location, office_day, office_hours, class_day, class_hours

c) **Teacher Entity**
   - teacher_ID → teacher_name, email, office_location

d) **Department Entity**
   - dep_ID → dep_name, office_location

e) **Tuition Entity**
   - student_ID, tuition_ID → item_name, due_date, item_amount

f) **Assignment Entity**
   - assignment_name → assignment_weight, assignment_due_date

g) **Enrollments Entity**
   - enrollment_ID → student_ID, course_ID

h) **Teaching Entity**
   - teaching_ID → teacher_ID, course_ID

i) **Faculty Entity**
   - fac_name → fac_budget, fac_address, fac_year_founded

6.2 Functional Dependencies for Relationships

a) **Owes Relation**
   - tuition_ID → student_ID

b) **Enrolled_In Relation**
   - Originally a M-N relationship; can't have any functional dependencies.
   - To remove this M-N relationship, we split up the Enrolled_in relation by introducing a third table called Enrollments, and two new relationships: Registered and Include.
   - **Enrollments Relation**
     - enrollment_ID → student_ID, course_ID
   - **Registered Relation**
     - student_ID → enrollment_ID
   - **Include Relation**

- course_ID → enrollment_ID

c) Recieves_Mark Relation

- assignment_ID → student_ID
- assignment_ID, student_ID → mark_received

d) Assigns Relation

- assignment_ID → course_ID

e) Offers Relation

- course_ID → dep_ID

f) Teaches Relation

- Originally a M-N relationship; can't have any functional dependencies.
- To remove this M-N relationship, we split up the Teaches relation by introducing a third table called Teaching.
- **Teaching Relation**
  - teaching_ID → course_ID, teacher_ID
- **Teaches_Course Relation**
  - teacher_ID → teaching_ID
- **Teaches Relation**
  - course_ID → teaching_ID

g) Manages Relation

- teacher_ID → dep_ID

h) Advises Relation

- dep_name → fac_name

7. VERIFICATION OF 3NF

Tables not originally in 1NF:

Student

- Composite attribute birth_date stored year, month, and day broke 1NF because they are not atomic attributes
- Removed and replaced the attribute birth_date with three new attributes: birth_year, birth_month, and birth_day

Tables not in 2NF:

- All tables were already 2NF, as all tables only had a single primary key.

Tables in 3NF:

Student is 3NF

- While non-primary key attribute email could derive the rest of the attributes, because email is unique per student_id, it does not cause a 3NF issue.

Teacher

- While non-primary key attribute email could derive the rest of the attributes, because email is unique per teacher_id, it does not cause a 3NF issue.

Tuition

- While non-primary key attribute item_name could derive attribute item_amount, because item_name is unique per tuition_id, it does not cause a 3NF issue.

Department

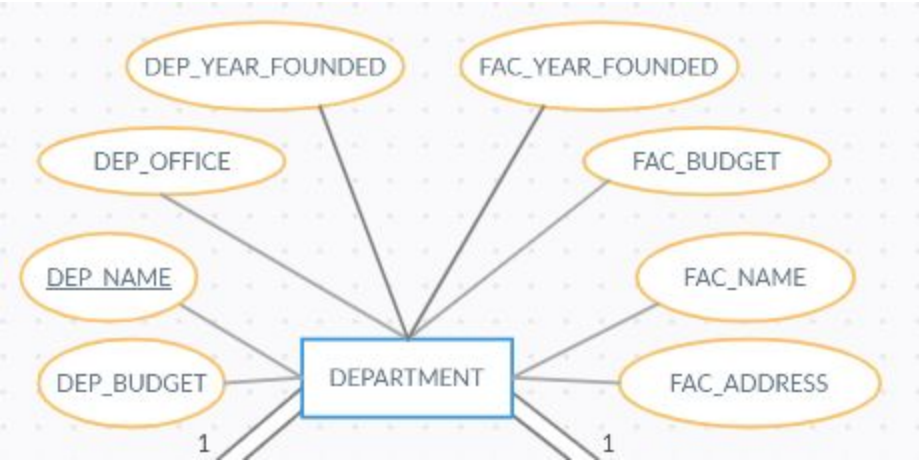
- While non-primary key attribute dep_name could derive attribute office_location, because dep_name is unique per dep_id, it does not cause a 3NF issue.
- However, we removed the primary key dep_id and replaced it with new primary key dep_name.

8. BERNSTEIN'S ALGORITHM FOR 3NF

Since our tables were already in 3NF, we had to add additional attributes and functional dependencies so we can apply the algorithms appropriately.

Step 1:

First we added six attributes to the department entity: dep_budget, dep_year_founded, fac_name, fac_budget, fac_address and fac_year_founded. The resulting entity is:



department(dep_name, dep_address, fac_name, fac_dean, fac_address).

FDs: {
dep_name → dep_office, dep_year_founded, dep_budget
dep_name → fac_name,
fac_name → fac_address, fac_budget, fac_year_founded }

Step 2a:

FDs: {

1. dep_name → dep_office,
2. dep_name → dep_year_founded,
3. dep_name → dep_budget,
4. dep_name → fac_name,
5. fac_name → fac_address,
6. fac_name → fac_budget,
7. fac_name → fac_year_founded }

Step 2b:

1. $\{dep\_name\}^+ =$

{dep_name, dep_year_founded, dep_budget, fac_name, fac_address, fac_budget, fac_year_founded}

dep_office is not in the closure, so 1 is not redundant

2. $\{dep\_name\}^+$ =
{dep_name, dep_office, dep_budget, fac_name, fac_address, fac_budget, fac_year_founded}

dep_year_founded is not in the closure, so 2 is not redundant,

3. $\{dep\_name\}^+$ =
{dep_name, dep_office, dep_year_founded, fac_name, fac_address, fac_budget, fac_year_founded}

dep_budget is not in the closure, so 3 is not redundant

4. $\{dep\_name\}^+$ =
{dep_name, dep_office, dep_year_founded, dep_budget, fac_address, fac_budget, fac_year_founded}

fac_name is not in the closure, so 4 is not redundant

5. $\{fac\_name\}^+$ =
{fac_budget, fac_year_founded}

fac_address is not in the closure, so 5 is not redundant

6. $\{fac\_name\}^+$ =
{fac_address , fac_year_founded}

fac_budget is not in the closure, so 6 is not redundant

7. $\{fac\_name\}^+$ =
{fac_address , fac_budget}

fac_year_founded is not in the closure, so 6 is not redundant

Step 2c:
Since there is only one attribute on the left hand side of every functional dependency, we can skip this substep.

Step 3:
FDs: {

8. dep_name → dep_office,
9. dep_name → dep_year_founded,
10. dep_name → dep_budget,
11. dep_name → fac_name,
12. fac_name → fac_address,
13. fac_name → fac_budget,
14. fac_name → fac_year_founded }

Attributes on RHS but not on LHS (these attributes will **not be in any key**):

- dep_office
- dep_year_founded
- dep_budget
- fac_address
- fac_budget
- fac_year_founded

Attributes on LHS but not on RHS (these attributes **will be in every key**):

- dep_name

The only attribute remaining is fac_name. This attribute appears both on the LHS and RHS. We will test if fac_name is a key if $\{fac\_name\}^+$ contains all the attributes of the relation:

$\{fac\_name\}^+$ =
{fac_address , fac_budget, fac_year_founded}

The resulting closure does not contain all the attributes of the relation meaning fac_name is not part of the key.

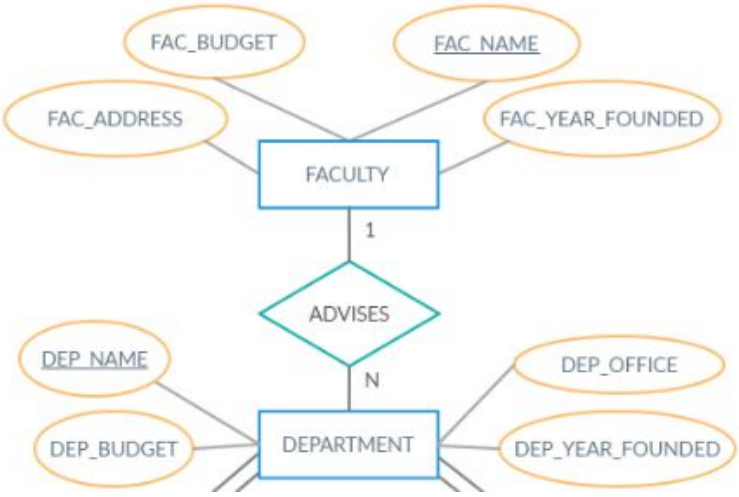
Therefore we have one candidate key: **{dep_name}**

Step 4:
Combine FDs with the same LHS:
FD1 = {dep_name → dep_office, dep_year_founded, dep_budget, fac_name}
FD2 = {fac_name → fac_address, fac_budget, fac_year_founded}

Resulting in the following relation schema:

- R1(dep_name, dep_office, dep_year_founded, dep_budget, fac_name)
- R2(fac_name, fac_address, fac_budget, fac_year_founded)

9. BCNF ALGORITHM



We return back to the original *department*(*dep_name, dep_address, fac_name, fac_dean, fac_address*) relation with FDs:

FDs: *{*
dep_name → dep_office, dep_year_founded, dep_budget, fac_name,
fac_name → fac_address, fac_budget, fac_year_founded
}

9.1 BCNF Algorithm

To determine whether this relation is in BCNF, we must determine whether each non-trivial, left-irreducible functional dependency is a candidate key as its determinant. To do this, we will find the closure of each determinant.

1. $\{dep\_name\}^+$ =
{dep_name, dep_office, dep_year_founded, dep_budget, fac_name, fac_address, fac_budget, fac_year_founded}

This determinant is a key as every attribute is found in the closure.

2. $\{fac\_name\}^+$ =
{fac_name, fac_address, fac_budget, fac_year_founded}

This determinant is not a key! As a result, *department* is not in BCNF with respect to *fac_name → fac_address, fac_budget, fac_year_founded*.

We set *department = R = Q =* {*dep_name, dep_address, fac_name, fac_dean, fac_address}*

We now decompose *R* into *R1* and *R2* through {*fac_name}*$^{+}$:

R1 = {dep_name, dep_office, dep_year_founded, dep_budget, fac_name}
R2 = {fac_name, fac_address, fac_budget, fac_year_founded}

We now derive the following FDs:

FD1 *= {dep_name → dep_office, dep_year_founded, dep_budget, fac_name}*
FD2 *= {fac_name → fac_address, fac_budget, fac_year_founded}*

Giving us keys ***{dep_name}*** for *R1* and ***{fac_name}*** for *R2*. As a result, R1 and R2 are now both in BCNF.

9.2 Functional Dependencies for Relationships

a) **Owes Relation**
   - tuition_ID → student_ID
   - Every nontrivial, left irreducible FD has a candidate key as its determinant, so therefore, this relation is in BCNF.

b) **Registered Relation**
   - student_ID → enrollment_ID
   - Every nontrivial, left irreducible FD has a candidate key as its determinant, so therefore, this relation is in BCNF.

c) **Include Relation**
   - course_ID → enrollment_ID
   - Every nontrivial, left irreducible FD has a candidate key as its determinant, so therefore, this relation is in BCNF.

d) **Recieves_Mark Relation**
   - assignment_ID, student_ID → mark_received

- Every nontrivial, left irreducible FD has a candidate key as its determinant, so therefore, this relation is in BCNF.

e) Assigns Relation

- assignment_ID → course_ID
- Every nontrivial, left irreducible FD has a candidate key as its determinant, so therefore, this relation is in BCNF.

f) Offers Relation

- course_ID → dep_ID
- Every nontrivial, left irreducible FD has a candidate key as its determinant, so therefore, this relation is in BCNF.

g) **Teaches_Course Relation**

- teacher_ID → teaching_ID
- Every nontrivial, left irreducible FD has a candidate key as its determinant, so therefore, this relation is in BCNF.

h) Teaches Relation

- course_ID → teaching_ID
- Every nontrivial, left irreducible FD has a candidate key as its determinant, so therefore, this relation is in BCNF.

i) Manages Relation

- teacher_ID → dep_ID
- Every nontrivial, left irreducible FD has a candidate key as its determinant, so therefore, this relation is in BCNF.

9.3 Functional Dependencies for Entities

a) Student Entity

- student_ID → birth_date, email, student_name, gpa, birth_date
- Every nontrivial, left irreducible FD has a candidate key as its determinant, so therefore, this relation is in BCNF.

b) Course Entity

- course_ID → course_name, course_location, office_day, office_hours, class_day, class_hours
- Every nontrivial, left irreducible FD has a candidate key as its determinant, so therefore, this relation is in BCNF.

c) Teacher Entity

- teacher_ID → teacher_name, email, office_location

- Every nontrivial, left irreducible FD has a candidate key as its determinant, so therefore, this relation is in BCNF.

d) Department Entity

- dep_ID → dep_name, office_location
- Every nontrivial, left irreducible FD has a candidate key as its determinant, so therefore, this relation is in BCNF.

e) Tuition Entity

- student_ID, tuition_ID → item_name, due_date, item_amount
- Every nontrivial, left irreducible FD has a candidate key as its determinant, so therefore, this relation is in BCNF.

f) Assignment Entity

- assignment_name → assignment_weight, assignment_due_date
- Every nontrivial, left irreducible FD has a candidate key as its determinant, so therefore, this relation is in BCNF.

g) Enrollments Entity

- enrollment_ID → student_ID, course_ID
- Every nontrivial, left irreducible FD has a candidate key as its determinant, so therefore, this relation is in BCNF.

h) Teaching Entity

- teaching_ID → teacher_ID, course_ID
- Every nontrivial, left irreducible FD has a candidate key as its determinant, so therefore, this relation is in BCNF.

10. RELATIONAL ALGEBRA

Table name notation: Tables with their names capitalized (e.g. Receives_Mark) represents the table. Tables written with their names in lowercase (e.g. receives_mark) represents their set of attributes (e.g. *student_ID, assignment_name, mark_received* for Receives_Mark).

1. List all the students that qualify for the Dean's List:

$$deansListStudents \leftarrow \rho_{Student(Deans\_List,\ GPA)}\left(\Pi_{student\_name,\ gpa}(\sigma_{GPA \geq 3.8}(Student))\right)$$

$$result \leftarrow {}^{F} ORDER\ BY\ GPA\ (deansListStudents)$$

2. List all the students (student name and student ID) that are enrolled in ART201:

$$studentsInART201 \leftarrow \sigma_{course\_ID="ART201"}(Student \bowtie Enrollments)$$

$$result \leftarrow \rho_{Student(ART201\_Students,\ student\_ID)}\left(\Pi_{student\_name,\ student\_ID}(studentsInART201)\right)$$

3. Show the average GPA of all students:

$$result \leftarrow {}^{F} AVERAGE\ gpa\ (Student)$$

4. For each course, list the course ID, course name, number of students in the course, and the number of teachers in the course:

$$tables \leftarrow \sigma_{Enrollments.course\_ID=Teaching.course\_ID \wedge Enrollments.course\_ID=Course.course\_ID \wedge Teaching.course\_ID=Course.course\_ID}((Enrollments \times Course) \times Teaching)$$

$$courses \leftarrow \Pi_{course\_ID,\ course\_name}\left[\Pi_{enrollments \cup teaching \cup course}(tables)\right]$$

$$R1(course\_ID,\ Number\_of\_Students) \leftarrow {}^{F} DISTINCT\ course\_ID\ (courses),\ COUNT\ student\_ID\ (DISTINCT\ student\_ID\ (tables))$$

$$R2(course\_ID,\ Number\_of\_Teachers) \leftarrow {}^{F} DISTINCT\ course\_ID\ (courses),\ COUNT\ teacher\_ID\ (DISTINCT\ teacher\_ID\ (tables))$$

$$result \leftarrow {}^{F} ORDER\ BY\ course\_name\ [GROUP\ BY\ course\_ID,\ course\_name\ ((courses \bowtie R1) \bowtie R2)]$$

5. List the average mark, minimum mark, and maximum mark for each assignment in ART201:

$$tables \leftarrow \sigma_{course\_ID = "ART201" \wedge Assigns.assignment\_name = Receives\_Mark.assignment\_name}(Assigns \times Receives\_Mark)$$

$$assignments \leftarrow \Pi_{assignment\_name}\left[\Pi_{assigns \cup receives\_mark}(tables)\right]$$

$$R1(assignment\_name,\ Average\_Mark) \leftarrow {}^{F}AVERAGE\ mark\_received\ (tables)$$

$$R2(assignment\_name,\ Lowest\_Mark) \leftarrow {}^{F}MIN\ mark\_received\ (tables)$$

$$R3(assignment\_name,\ Highest\_Mark) \leftarrow {}^{F}MAX\ mark\_received\ (tables)$$

$$result \leftarrow {}^{F}GROUP\ BY\ assignment\_name\ (assignments \bowtie R1 \bowtie R2 \bowtie R3)$$

6. For each course, list the course ID and course name, and list all the assignments in that course and list the average mark, minimum mark, and maximum mark for each assignment:

$$tables \leftarrow \sigma_{Assigns.course\_ID = Course.course\_ID \wedge Assigns.assignment\_name = Receives\_Mark.assignment\_name}((Course \times Assigns) \times Receives\_Mark)$$

$$courses \leftarrow \Pi_{course\_ID,\ course\_name,\ assignment\_name}\left[\Pi_{course \cup assigns \cup receives\_mark}(tables)\right]$$

$$R1(assignment\_name,\ Average\_Mark) \leftarrow {}^{F}AVERAGE\ mark\_received\ (tables)$$

$$R2(assignment\_name,\ Lowest\_Mark) \leftarrow {}^{F}MIN\ mark\_received\ (tables)$$

$$R3(assignment\_name,\ Highest\_Mark) \leftarrow {}^{F}MAX\ mark\_received\ (tables)$$

$$result \leftarrow {}^{F}ORDER\ BY\ course\_name\ [GROUP\ BY\ course\_ID,\ course\_name,\ assignment\_name\ (courses \bowtie R1 \bowtie R2 \bowtie R3)]$$

7. List all the teacher ID's, teacher names, teacher emails, and list all the classes they teach and the number of students in those classes:

$$tables \leftarrow \sigma_{Teacher.teacher\_ID=Teaching.teacher\_ID \wedge Teaches.course\_ID=Enrollments.course\_ID}((Teacher \times Teaching) \times Enrollments)$$

$$teachers \leftarrow \Pi_{teacher\_ID,\ teacher\_name,\ email,\ course\_ID}\left[\Pi_{teacher \cup teaching \cup enrollments}(tables)\right]$$

$$R1(course\_ID,\ Number\_of\_Students) \leftarrow {}^{F}\ DISTINCT\ course\_ID\ (teachers),\ COUNT\ student\_ID\ (DISTINCT\ student\_ID\ (teachers))$$

$$result \leftarrow {}^{F}\ ORDER\ BY\ teacher\_name\ [GROUP\ BY\ teacher\_ID,\ teacher\_name,\ email,\ course\_ID\ (teachers \bowtie R1)]$$

8. For every department, list the department id, department name, office location, number of teachers, number of courses, and number of students in each department:

$$tables \leftarrow \sigma_{Department.dep\_ID=Manages.dep\_ID \wedge Department.dep\_ID=Offers.dep\_ID \wedge Offers.course\_ID=Enrollments.course\_ID}(((Department \times Manages) \times Offers) \times Enrollments)$$

$$departments \leftarrow \Pi_{dep\_ID,\ dep\_name,\ office\_location}\left[\Pi_{department \cup manages \cup offers \cup enrollments}(tables)\right]$$

$$R1(dep\_ID,\ Number\_of\_Teachers) \leftarrow {}^{F}\ DISTINCT\ dep\_ID\ (departments),\ COUNT\ teacher\_ID\ (DISTINCT\ teacher\_ID\ (tables))$$

$$R2(dep\_ID,\ Number\_of\_Courses) \leftarrow {}^{F}\ DISTINCT\ dep\_ID\ (departments),\ COUNT\ course\_ID\ (DISTINCT\ course\_ID\ (tables))$$

$$R3(dep\_ID,\ Number\_of\_Students) \leftarrow {}^{F}\ DISTINCT\ dep\_ID\ (departments),\ COUNT\ student\_ID\ (DISTINCT\ student\_ID\ (tables))$$

$$result \leftarrow {}^{F}\ ORDER\ BY\ dep\_name\ [GROUP\ BY\ dep\_ID,\ dep\_name,\ office\_location\ (departments \bowtie R1 \bowtie R2 \bowtie R3)]$$

9. Display all teachers that teach a computer science course (starts with CPS) OR an art course (starts with ART):

$$result \leftarrow \Pi_{cs\_teachers}(CS\_Teachers)$$

10. Display count of students, courses, teachers, departments, and assignments:

$$result \leftarrow \Pi_{table\_counts}(Table\_Counts)$$

11. For each student, list their student ID, student name, email, and calculate the total amount of their tuition:

$$result \leftarrow \Pi_{total\_tuitions}(Total\_Tuitions)$$

12. Displays all teachers that are teaching a course, alongside the course that they are teaching:

$$tables \leftarrow \sigma_{Teaching.teacher\_ID=Teacher.teacher\_ID \wedge Teaching.course\_ID=Course.course\_ID}((Teaching \times Teacher) \times Course)$$

$$teachers \leftarrow \Pi_{teacher\_name,\ course\_name}\left[\Pi_{teaching \cup teacher \cup course}(tables)\right]$$

$$result \leftarrow {}^{F}ORDER\ BY\ \ teacher\_name\ (teachers)$$

13. List all teachers that are not from the science department:

$$tables \leftarrow \sigma_{dep\_ID="740937523"}(Manages \times Teacher)$$

$$teachers \leftarrow \Pi_{Manages.teacher\_ID,\ Teacher.teacher\_name,\ Teacher.email,\ Teacher.office\_location}(tables)$$

$$result \leftarrow \Pi_{teacher\_ID,\ teacher\_name,\ email,\ office\_location}(Teacher) - teachers$$

11. CONCLUSION

Creating this university course management system taught us the fundamentals of both designing and implementing a database. We learned how to design an ER model, create tables, design views, writing both simple and advanced queries, derive functional dependencies, and normalize tables to both 3NF and BCNF.